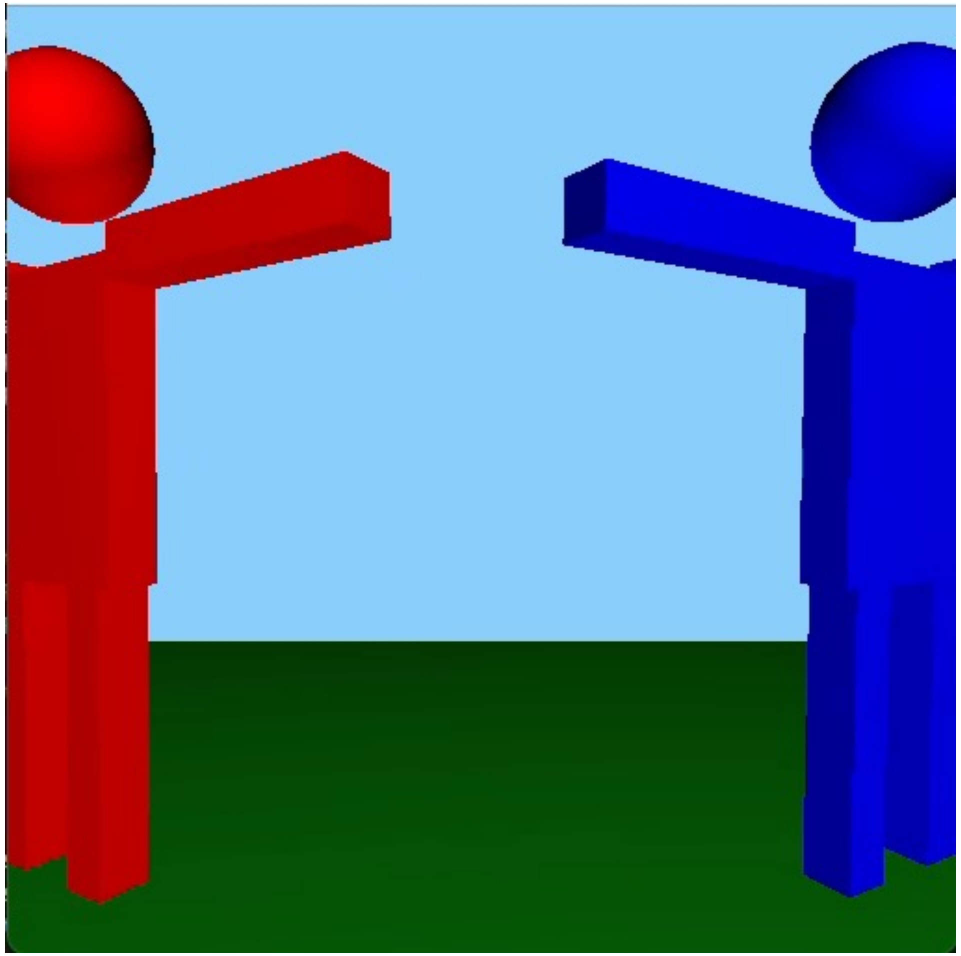
CS380: Introduction to Computer Graphics

Lab Session 5

EUNJI HONG

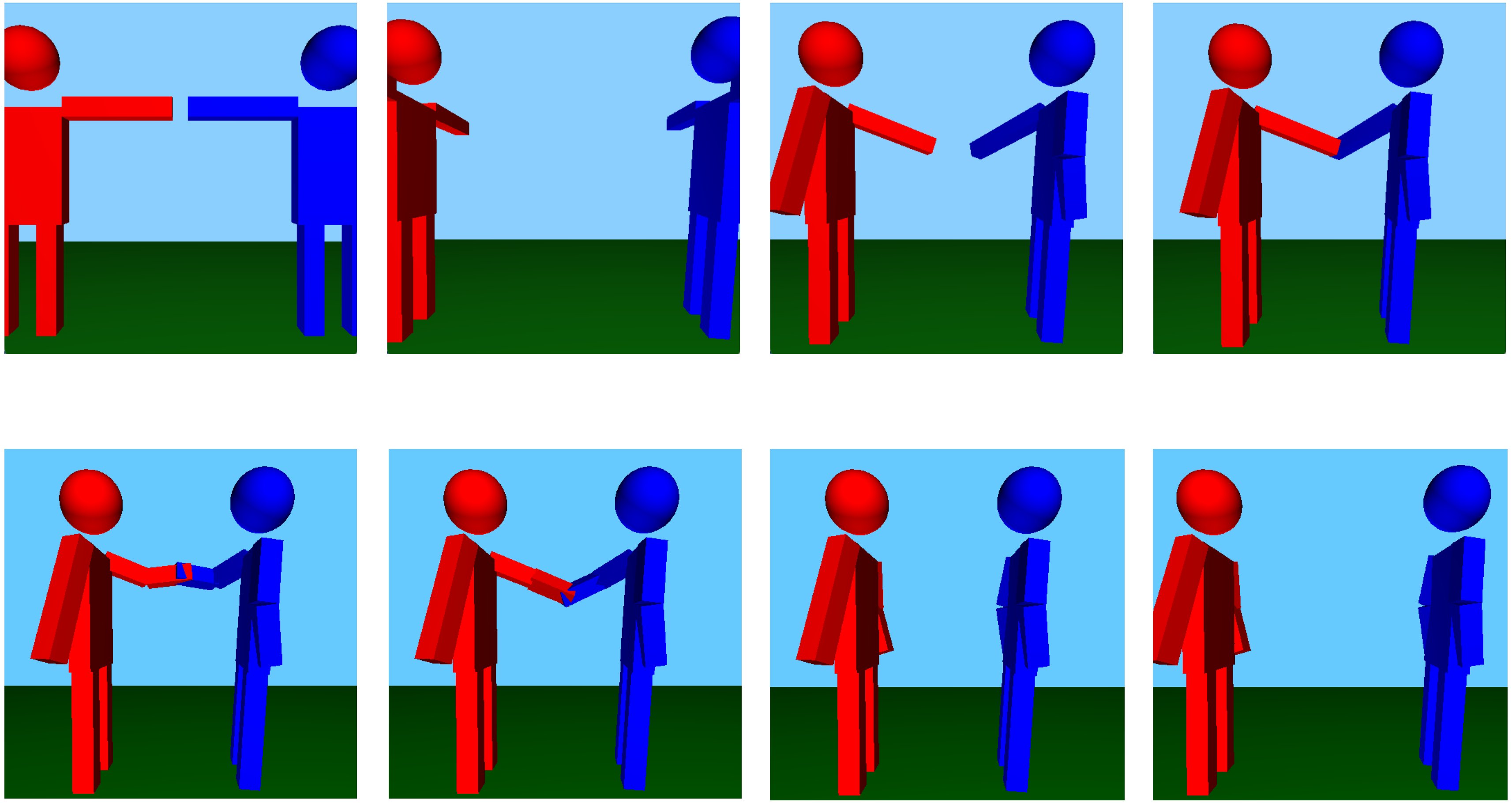
Spring 2023
KAIST

Keyframe Animation



Keyframe Animation

악수함



Assignment 5: Keyframe Animation I

- **Task 1**: Managing a keyframe list.
- **Task 2**: Linearly interpolating rigid body transformations across the keyframes.
- **Task 3**: Playing animation by interpolating the keyframes.

Setting Up Assignment 5

- Assignment 5 will be built upon your assignment 4 codebase.
- Download the assignment 5 code files and integrate them with your assignment 4 codebase.

Setting Up Assignment 5

These are the files for this assignment.

- sgutils.h contains the scene graph utility functions.
- interpolation.h includes the TODO’s for interpolation (Task 2).

We provide an additional tutorial code about C++ list structure.
You don’t need to include this file in your submission.

- list_tutorial.cpp contains a tutorial for the list container in C++ that can be used for implementing keyframe list structure.

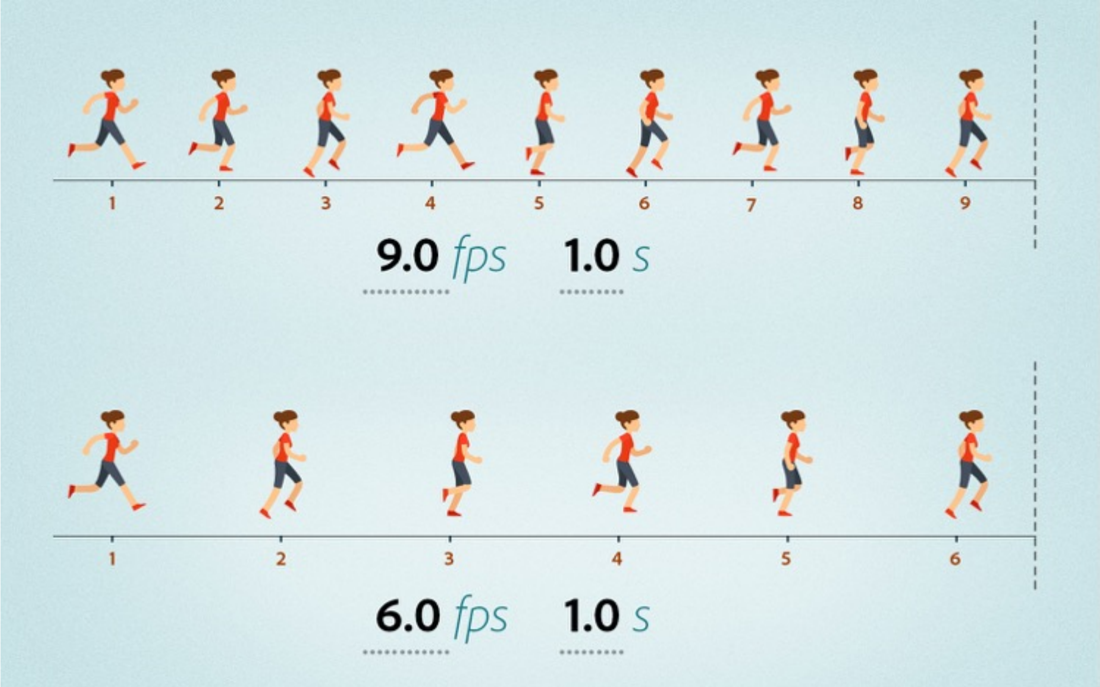
Animation

- Animation is a sequence of frames.
- Frame is the state of the scene at a specific moment in time.

Slide page credit: KAIST CS380 (Spring 2021) – Lab 6

Animation

A higher number of frames per second results in smoother motion but also leads to increased memory consumption.



https://helpx.adobe.com/animate/using/time.html

Keyframe Animation

- Stores a few “key” frames and interpolates the frames in-between the keyframes.
- Provides smooth motion while consuming less memory compared to storing all the frames directly.

Task 1: Managing a Keyframe List

Task 1: Keyframe List

- Implement a function to extract and store a frame at a specific instant in time, which consists of one RBT for each `SgRbtNode` in the scene graph.

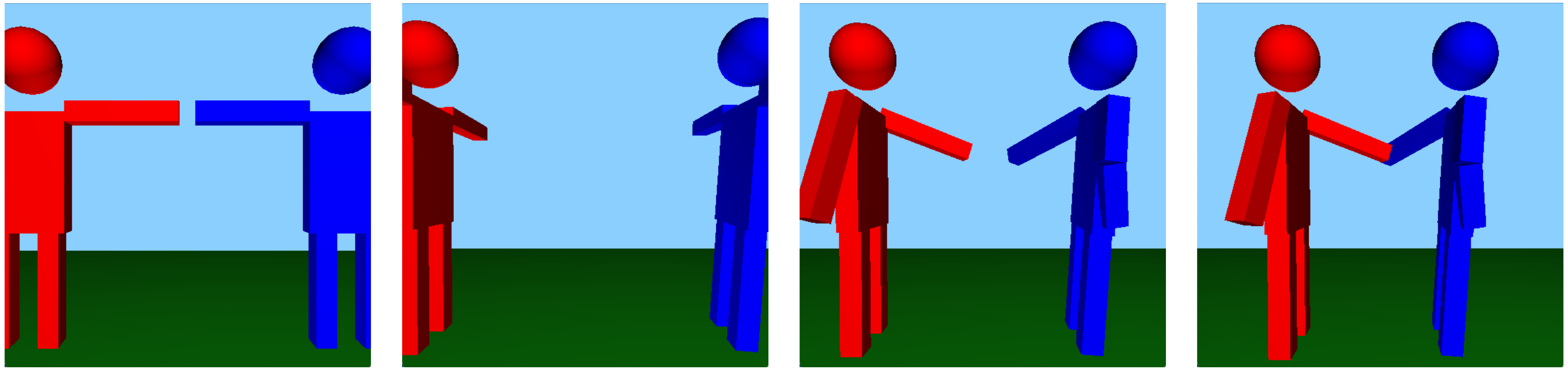
Keyframe List

- Implement a function to apply a stored frame to the scene graph, enabling the display of the frame's state on the screen.
- Implement a keyframe list class that can fully support all the required hotkey actions.

Task 1: Keyframe List Hotkeys

- Space(ASCII 32): Display the current keyframe.
- ‘u’: Update the current keyframe.
- ‘>’: Advance to the next keyframe.
- ‘<’: Go back to the previous keyframe.
- ‘d’: Delete the current keyframe.
- ‘n’: Create a new keyframe.
- ‘i’: Import (read) a keyframe list file.
- ‘w’: Export (write) a keyframe list file.

Keyframe List



Keyframe List

Keyframe 0
- Scene - RigTform
- Ground - RigTForm
- Robot 1 - RigTForm
- ⋮

Keyframe 1
- Scene - RigTform
- Ground - RigTForm
- Robot 1 - RigTForm
- ⋮

Keyframe 2
- Scene - RigTform
- Ground - RigTForm
- Robot 1 - RigTForm
- ⋮

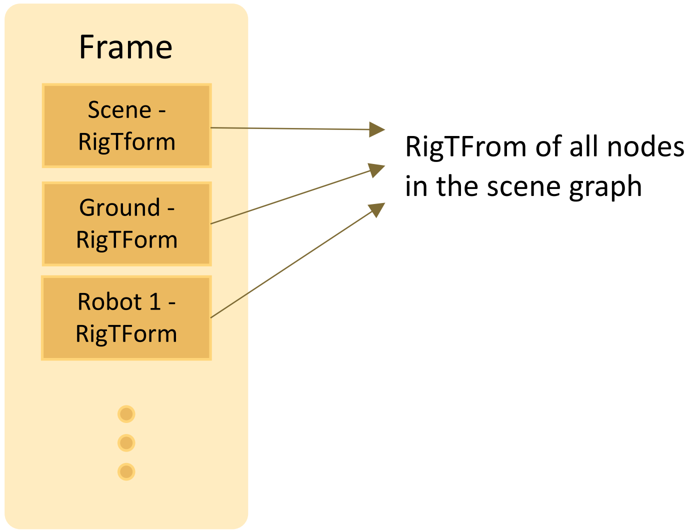
Keyframe 3
- Scene - RigTform
- Ground - RigTForm
- Robot 1 - RigTForm
- ⋮

…

Frame

- A frame is a set of RigTForms for the transformation nodes in the scene graph.
- You can represent a frame using any data structure, but we recommend using a vector, such as std::vector<RigTForm>.

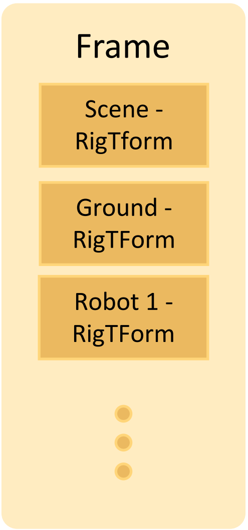
frame is vector of RigTForm

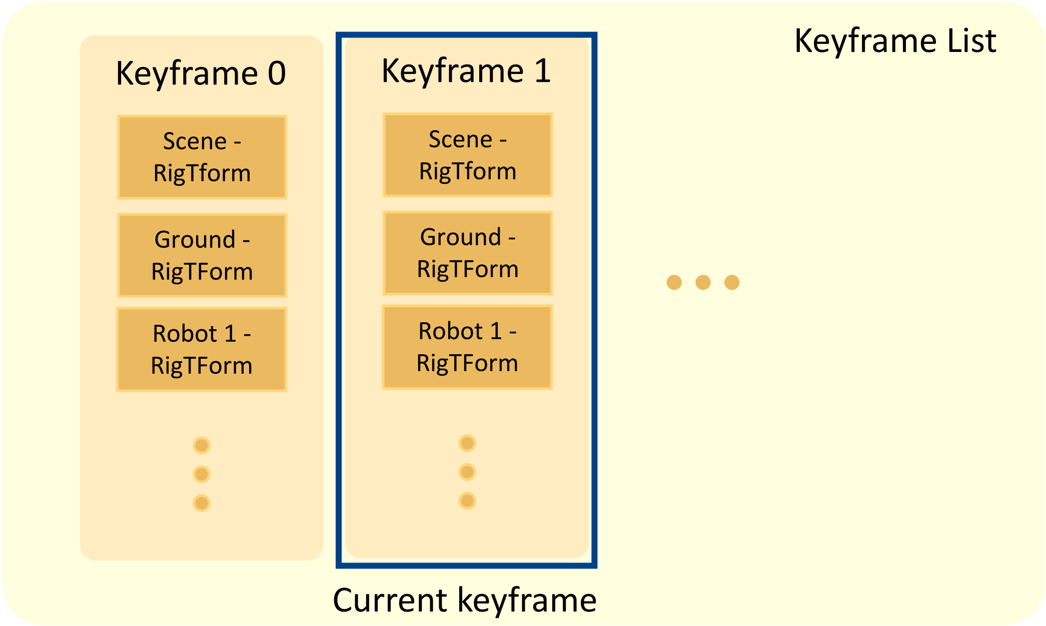


How to extract a frame from the scene graph will be explained later.

Keyframe List

- A keyframe list is a list of frames at each key instant.
- For editing purposes, at any given time, a keyframe list should have a variable to represent which keyframe is the current frame.





Keyframe List

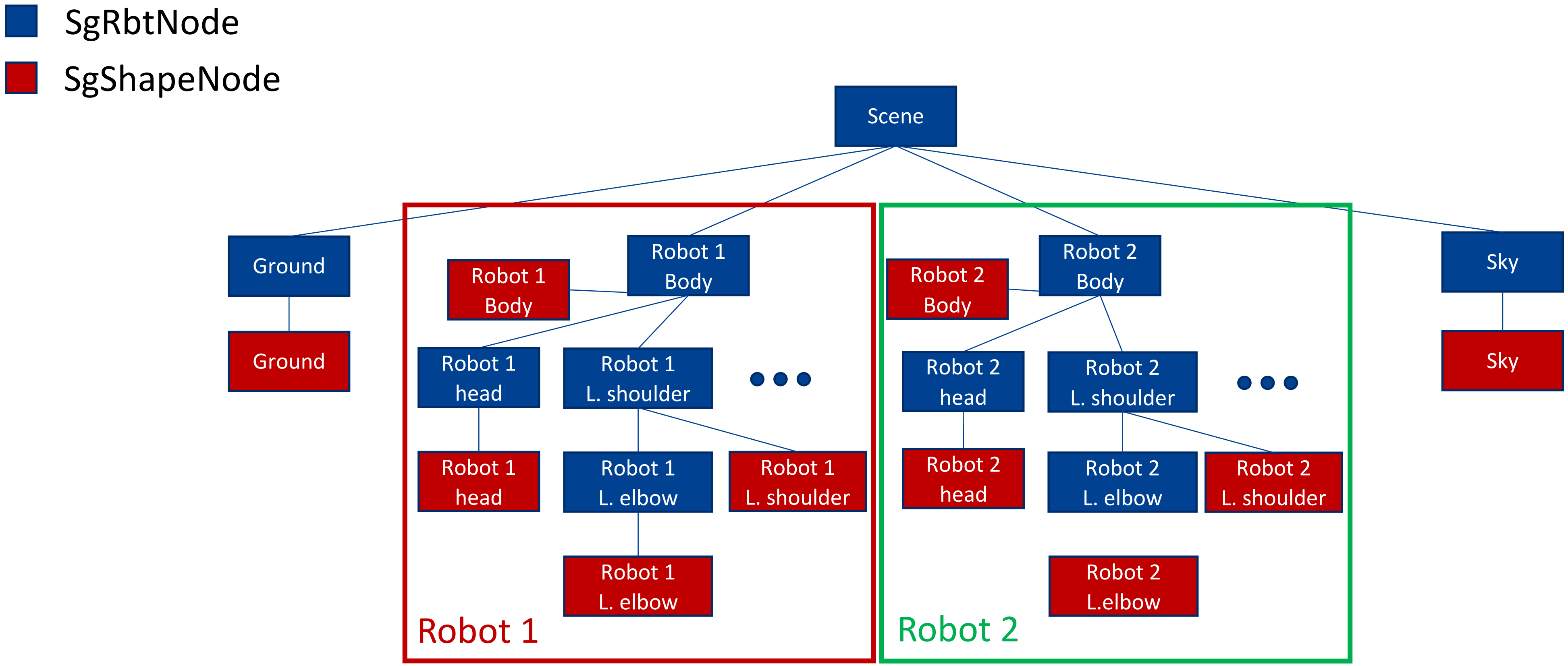
- When specific hotkeys are pressed, the corresponding action for the keyframe list should occur, such as adding a new keyframe or deleting the current keyframe.
- To easily insert and delete frames in the middle, we recommend using List from the C++ STL.
- Please refer to the specification's appendix, the tutorial code or the extra pages for more information on C++ STL list.
  - https://www.simplilearn.com/tutorials/cpp-tutorial/cpp-list
  - https://thispointer.com/c-different-ways-to-iterate-over-a-list-of-objects/

List Tutorial

We provide a tutorial code list_tutorial.cpp to help you utilize the C++ list structure which contains the examples of list functions. You don't need to include this file in your assignment submission.

```cpp
int main(){
    // Create an empty list of integers
    std::list<int> l1; // empty list of ints
    std::list<int> l2 = std::list<int>(); // empty list of ints


    // Create a list of integers with initial values
    std::list<int> l3 = {1, 2, 3, 4, 5}; // list initialized to values 1-5
    std::list<int> l4 = std::list<int>(5, 77); // list initialized to 5 values of 1
    std::list<int> l5 = std::list<int>(l3); // list initialized to a copy of l3



    // Add elements to the list
    // - list.push_back(value) : add value to the end of the list
    l1.push_back(1); // l1 = {1}
    l1.push_back(2); // l1 = {1, 2}
    l1.push_back(3); // l1 = {1, 2, 3}

    // - list.push_front(value) : add value to the front of the list
    l1.push_front(4); // l1 = {4, 1, 2, 3}

    // - list.insert(iterator, value) : add value before the iterator
    l2.insert(l2.begin(), 1); // l2 = {1}
    l2.insert(l2.begin(), 2); // l2 = {2, 1}
    l2.insert(l2.begin(), 3); // l2 = {3, 2, 1}
```

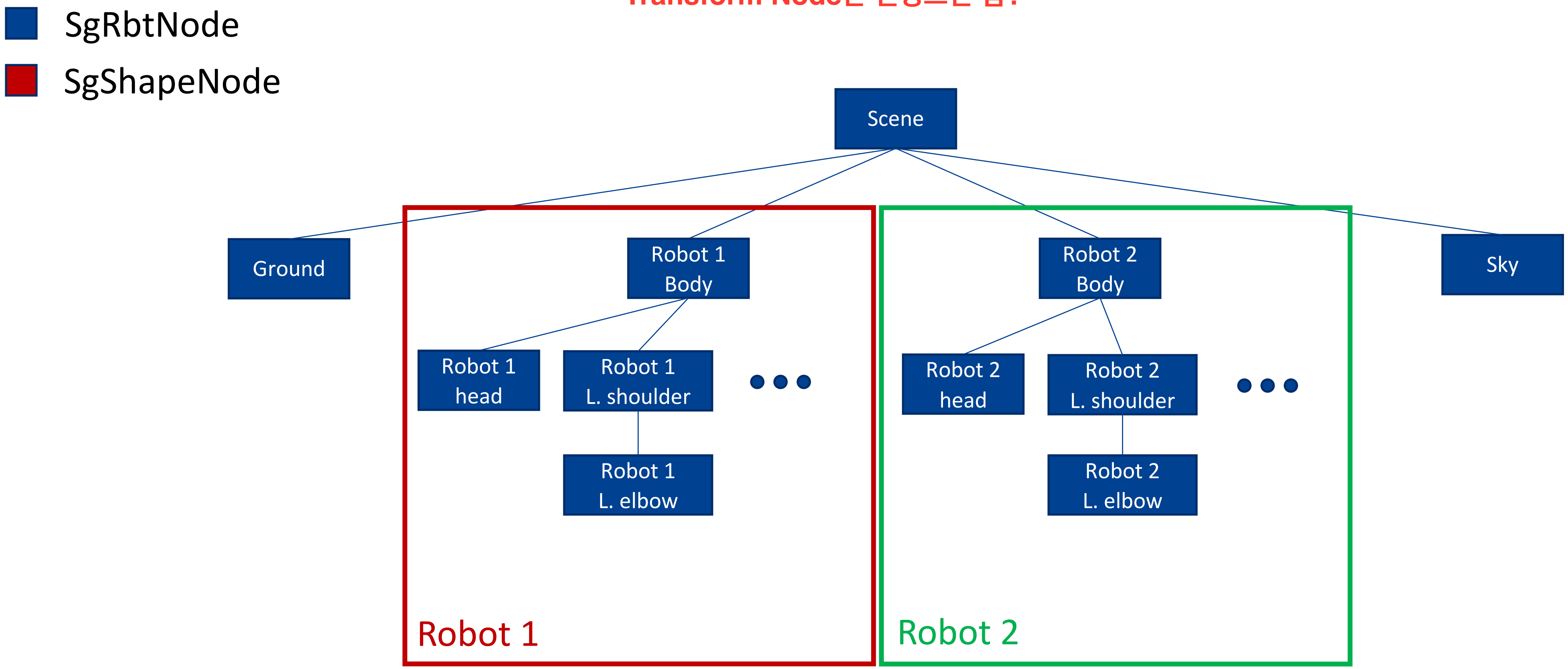
Recap Assignment 4: Scene Graph



Slide page credit: Hyunjin Kim

Scene Graph Transform Nodes

Transform Node만 신경쓰면 됨?

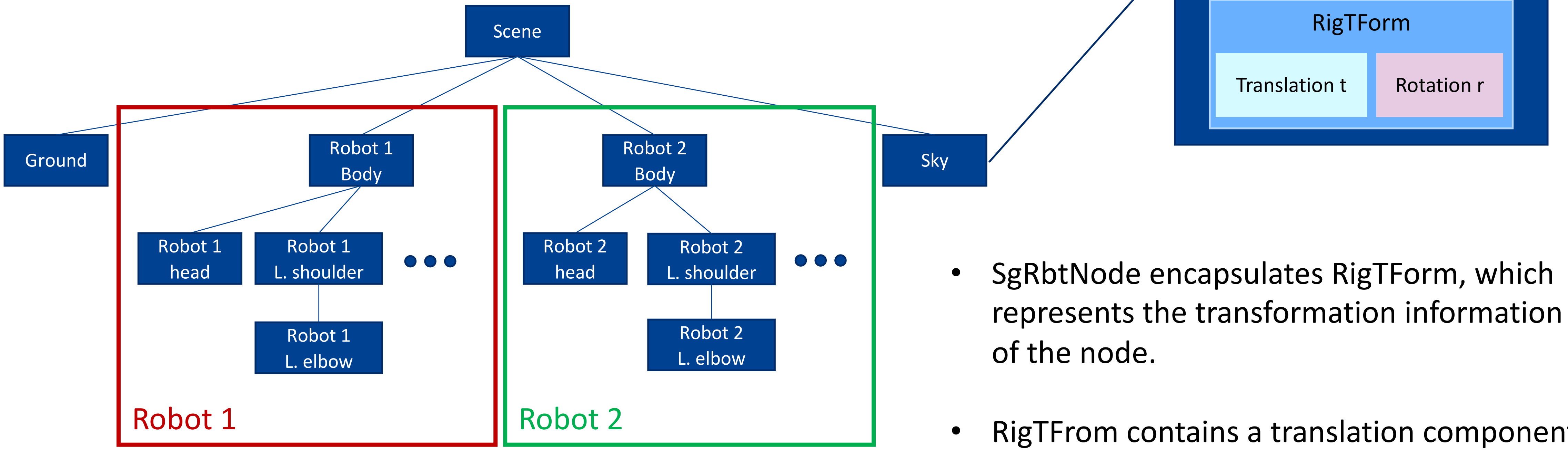


Slide page credit: Hyunjin Kim

Scene Graph Transform Nodes

■ SgRbtNode

tree에서 vector of frame을 어떻게 만들지?



- SgRbtNode encapsulates RigTForm, which represents the transformation information of the node.
- RigTFrom contains a translation component (t) and a rotation component (r).

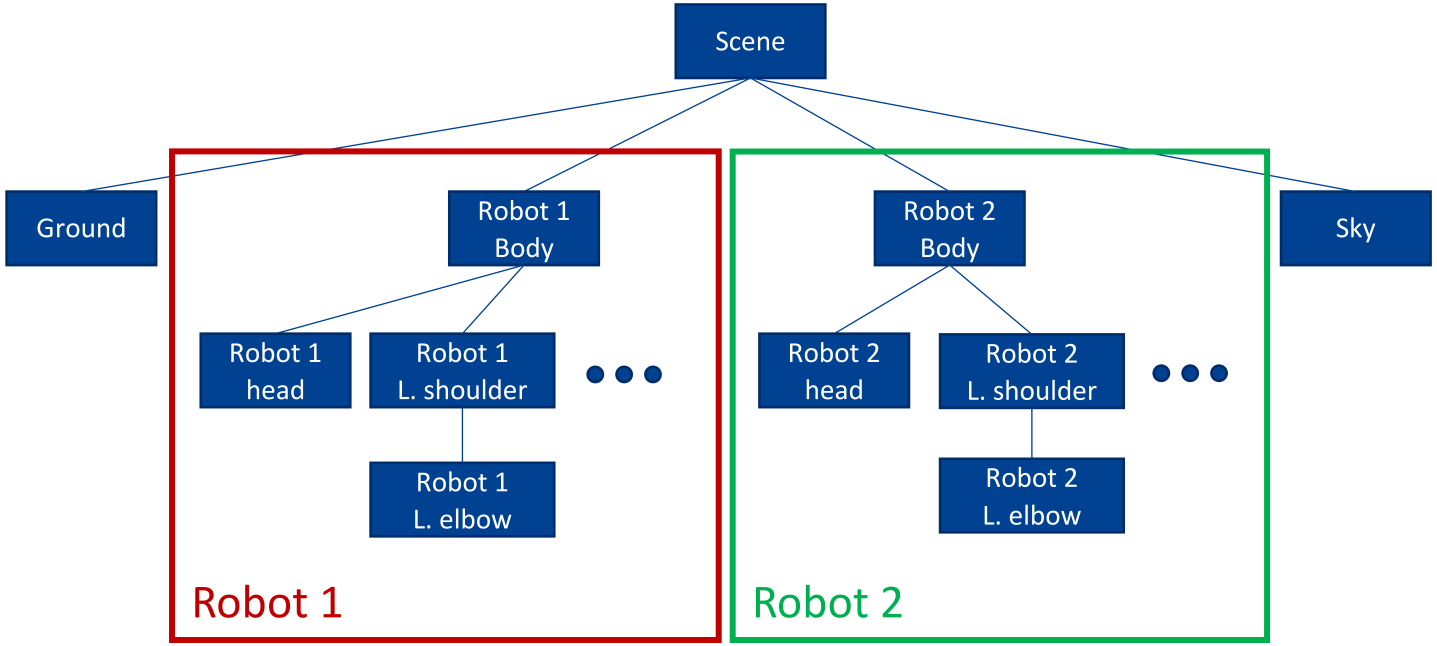
Slide page credit: Hyunjin Kim

Frame from the Scene Graph

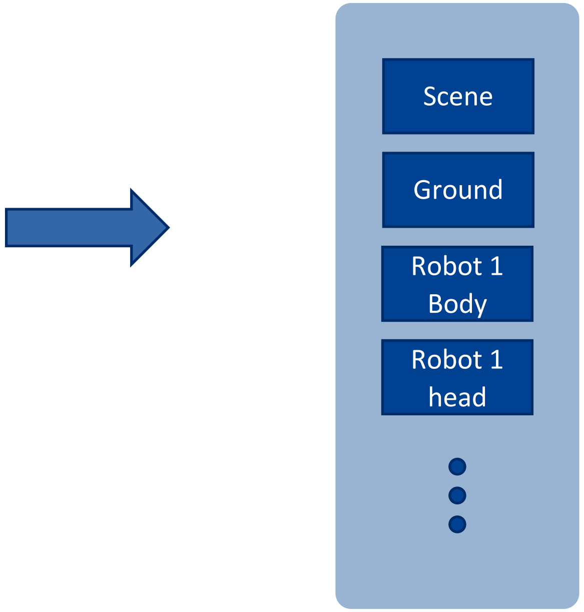
- As mentioned earlier, a frame is a vector of RigTForm for all nodes in the scene graph.
- To efficiently move data between a frame and the scene graph, it is beneficial to maintain a vector of node-points that point back to the corresponding SgRbtNode.
- That is, the *i*-th entry in the vector is a shared_ptr<SgRbtNode> that points to the *i*-th SgRbtNode in the scene graph.
- For this purpose, we provide a helper function called dumpSgRbtNodes.

vector of point of SbRbtNode

Scene Graph Transform Nodes

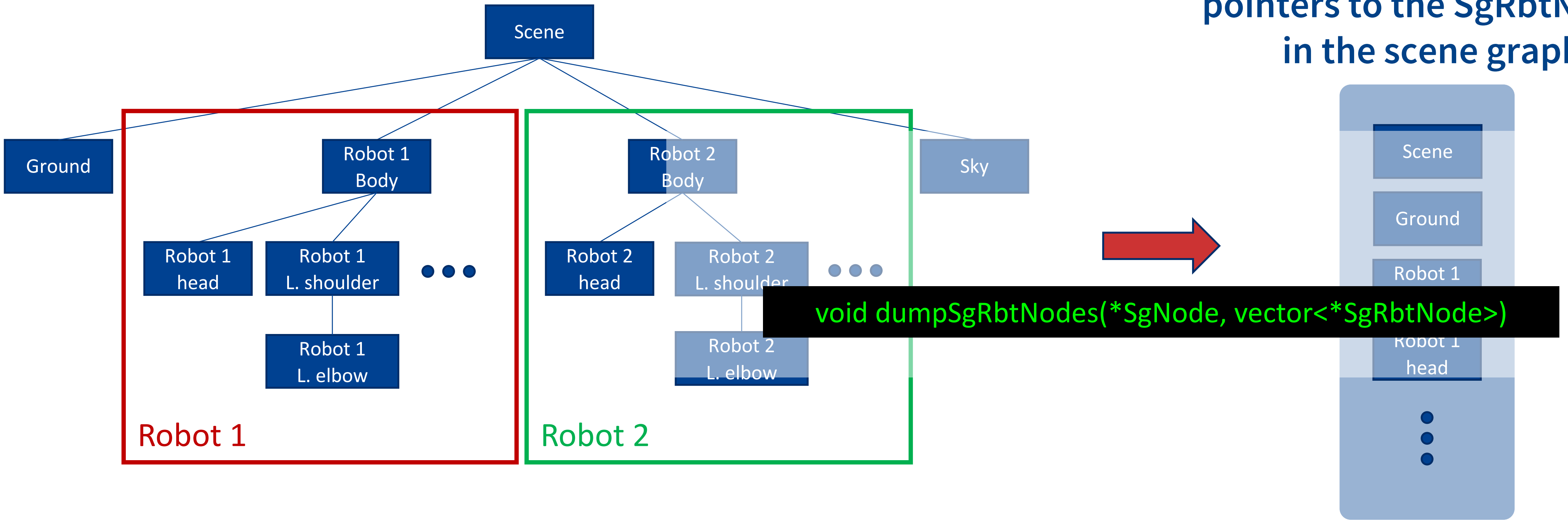


A vector containing pointers to the SgRbtNodes in the scene graph



Scene Graph Transform Nodes

A vector containing pointers to the SgRbtNodes in the scene graph



Scene Graph Transform Nodes

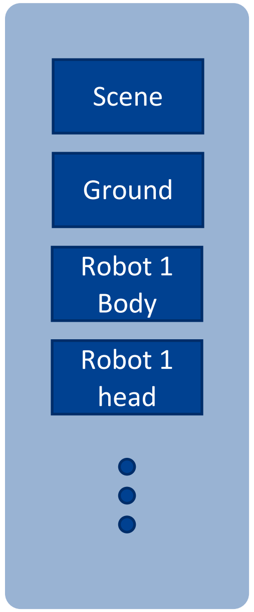
sgutils.h

```
inline void dumpSgRbtNodes(std::shared_ptr<SgNode> root, std::vector<std::shared_ptr<SgRbtNode> >& rbtNodes) {
  RbtNodesScanner scanner(rbtNodes);
  root->accept(scanner);
}
```

void dumpSgRbtNodes (*SgNode, vector<*SgRbtNode>)

- *SgNode: the pointer of the root node of the scene graph
- vector<*SgRbtNode>: an empty vector to be filled

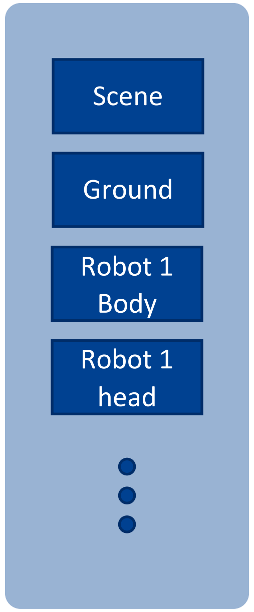
⇒ Fill the pointers of all SgRbtNodes in the scene graph in the given vector.



std::vector<*SgRbtNode>

Scene Graph Transform Nodes

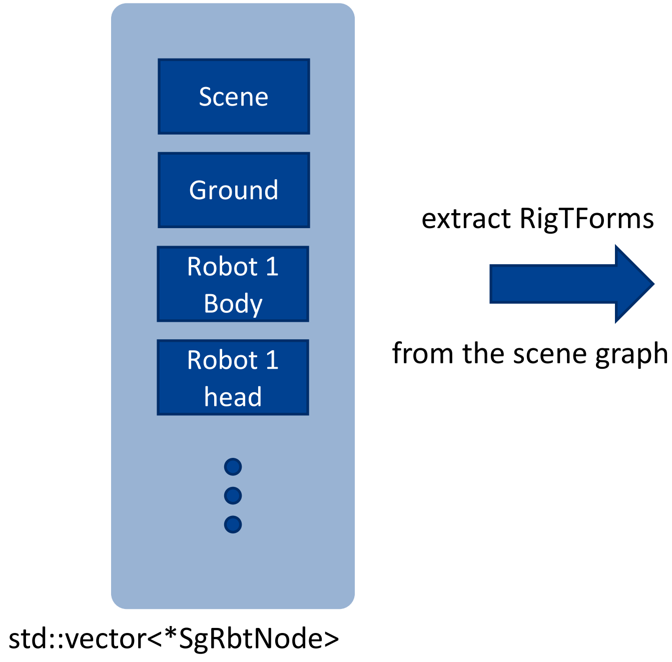
You can obtain a vector containing pointers to the transformation nodes in the scene graph by calling the function dumpSgRbtNodes once.



std::vector<*SgRbtNode>

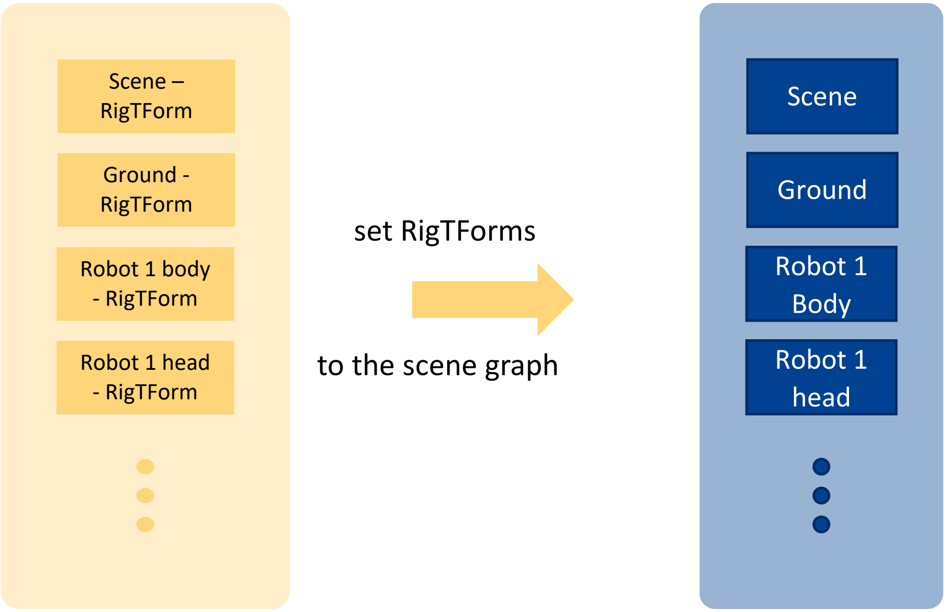
Scene Graph Transform Nodes

To extract a frame from the scene graph, you need to iterate through the scene graph pointer vector, obtain the RigTForm of each node, and store them in a vector (or another data structure of your choice).



Current Keyframe to the Scene Graph

You also need to apply the RigTForms from a frame to the scene graph. In this case, you will also use the vector containing pointers to the scene graph nodes.

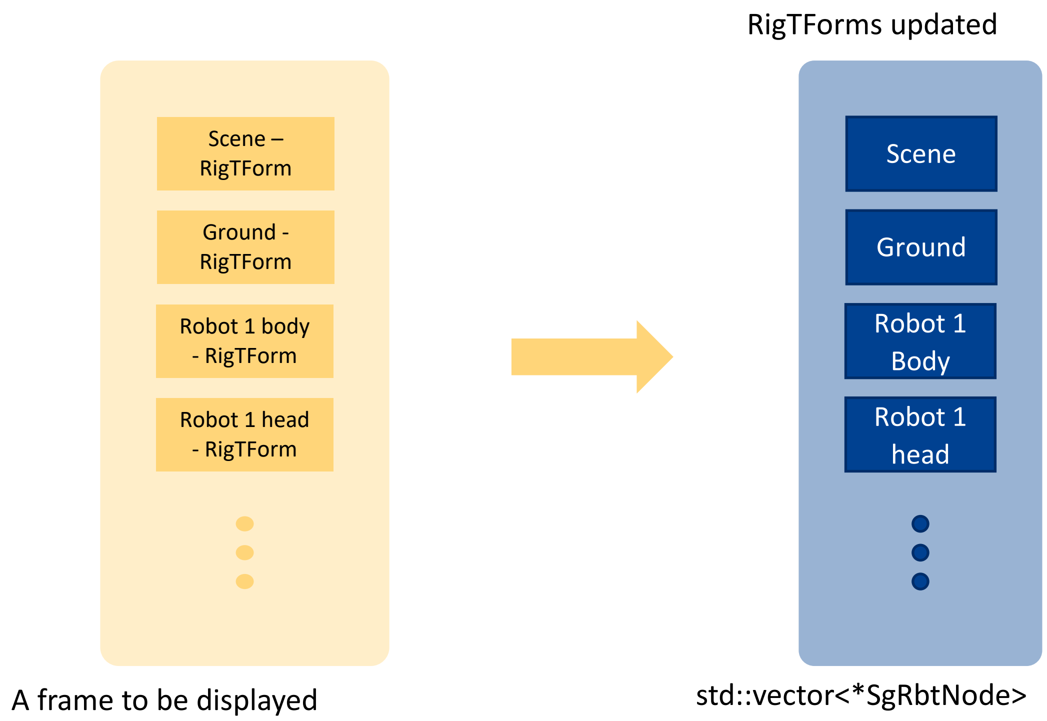


A frame to be displayed

std::vector<*SgRbtNode>

Current Keyframe to the Scene Graph

After applying the frame to the scene graph, you need to call glutPostRedisplay() to display the updated frame on the screen.



Keyframe List File I/O

- It is required to write and read a file containing the keyframe list.
- When pressing ‘w’, the current keyframe list should be saved to a file.
- When pressing ‘i’, if a keyframe list file exists, the current keyframe list should be replaced with the keyframes from the file.

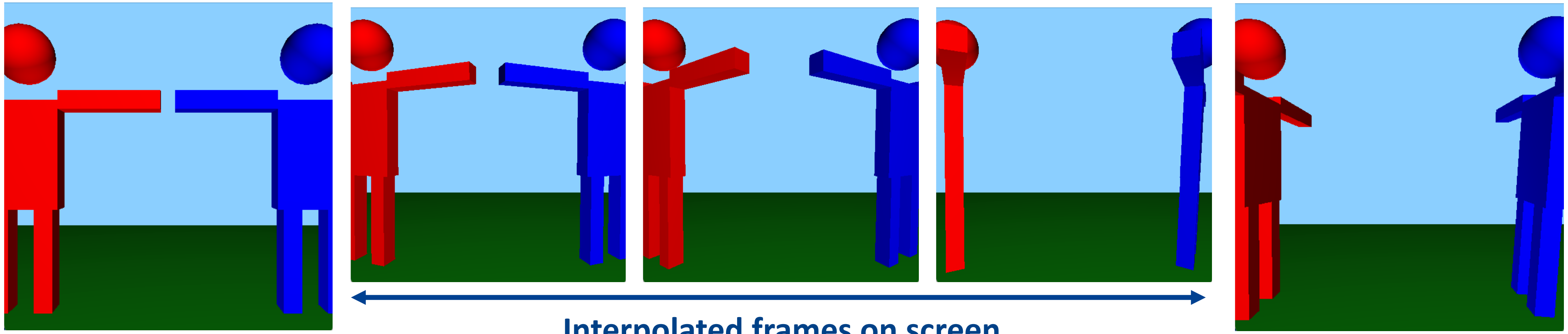
Keyframe List File I/O

- You can use any file I/O methods available in C++.
- You can save the keyframe list file in any format, such as .txt, etc.
- Please refer to the following C++ tutorial page for more information on file IO: https://cplusplus.com/doc/tutorial/files/

Task 2: Linearly Interpolating Rigid Body Transformations Across the Keyframes

key frame 사이 interpolation

Task2: Linear Interpolation



Interpolated frames on screen

Keyframe i
Scene - RigTform
Ground - RigTForm
Robot 1 - RigTForm

Scene - RigTform
Ground - RigTForm
Robot 1 - RigTForm

Scene - RigTform
Ground - RigTForm
Robot 1 - RigTForm

Scene - RigTform
Ground - RigTForm
Robot 1 - RigTForm

Keyframe i+1
Scene - RigTform
Ground - RigTForm
Robot 1 - RigTForm

Interpolated frames (RigTForm) in list

Task 2: Linear Interpolation

- Complete the TODOs in the `interpolation.h` file.
- Implement linear interpolation between two keyframes.
- Separately process the translation component and rotation component.

```
interpolate_rbt.t = lerp(rbt1.t, rbt2.t, alpha);
interpolate_rbt.r = slerp(rbt1.r, rbt2.r, alpha);
```

Slide page credit: KAIST CS380 (Spring 2021) – Lab 6

Linear Interpolation

- (Cvec3) Translation vector interpolation: simple vector linear interpolation (LERP)

$$\mathrm{lerp}(\mathbf{c}_0, \mathbf{c}_1, \alpha) := (1-\alpha)\mathbf{c}_0 + \alpha\mathbf{c}_1$$

- (Quat) Rotation quaternion interpolation: quaternion interpolation (SLERP)

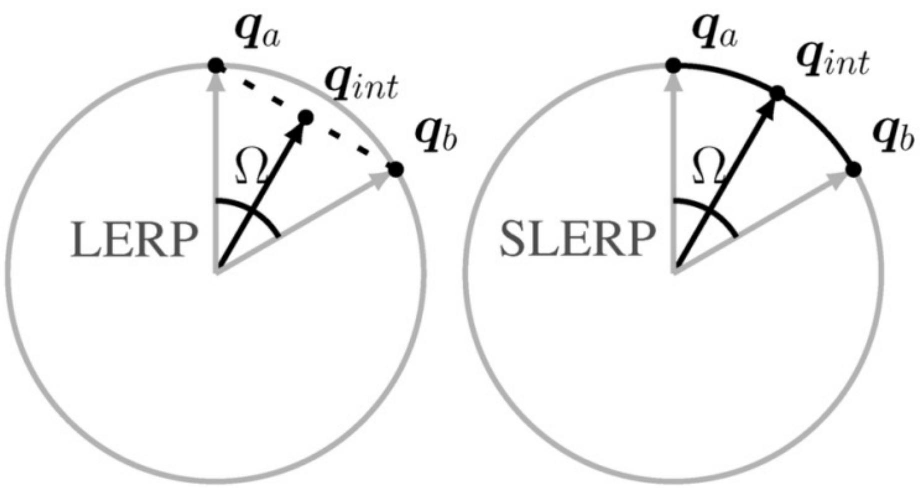
$$\mathrm{slerp}(\mathbf{q}_0, \mathbf{q}_1, \alpha) := (\mathrm{cn}(\mathbf{q}_1\mathbf{q}_0{}^{-1}))^{\alpha}\mathbf{q}_0$$

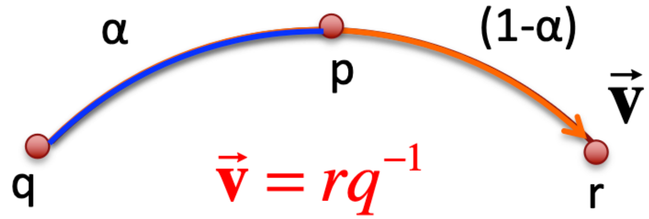
cn: conditional negate

Slide page credit: KAIST CS380 (Spring 2021) – Lab 6

Recap Lecture 9: SLERP

While linear interpolation (LERP) is performed on a line, we want to interpolate rotations on a sphere in the quaternion space using Spherical linear interpolation (SLERP).





$$p = \vec{\mathbf{v}}q = rq^{-1}q \qquad p = \vec{\mathbf{v}}^{\alpha}q = \left(rq^{-1}\right)^{\alpha} q$$

$$\vec{\mathbf{v}} = \begin{bmatrix} \cos\left(\frac{\theta}{2}\right) \\ \sin\left(\frac{\theta}{2}\right)\hat{\mathbf{k}} \end{bmatrix} \qquad \vec{\mathbf{v}}^{\alpha} = \begin{bmatrix} \cos\left(\frac{\alpha\theta}{2}\right) \\ \sin\left(\frac{\alpha\theta}{2}\right)\hat{\mathbf{k}} \end{bmatrix}$$

Recap Lecture 9: Power-Based Method

- Power-based Method

$$R_\alpha := (R_1 R_0^{-1})^\alpha R_0$$

- Power of quaternion $\vec{v}$

$$\vec{\mathbf{v}} = \begin{bmatrix} \cos\left(\frac{\theta}{2}\right) \\ \sin\left(\frac{\theta}{2}\right)\hat{\mathbf{k}} \end{bmatrix} \qquad \vec{\mathbf{v}}^\alpha = \begin{bmatrix} \cos\left(\frac{\alpha\theta}{2}\right) \\ \sin\left(\frac{\alpha\theta}{2}\right)\hat{\mathbf{k}} \end{bmatrix}$$

Recap Lecture 9: Power-Based Method

- Quaternions q and –q represent the same rotation. To obtain a shorter interpolation of less than 180 degrees, it is better to select the "short" interpolation.
- To achieve this, you can conditionally negate the quaternion. Before applying the power operator, first check the sign of the first coordinate and, if necessary, conditionally negate the quaternion to ensure the shorter interpolation.

$$\mathrm{slerp}(\mathbf{q}_0, \mathbf{q}_1, \alpha) := (\mathrm{cn}(\underline{\mathbf{q}_1 \mathbf{q}_0{}^{-1}}))^{\alpha}\mathbf{q}_0$$

cn: check the sign and negate
if the first coordinate is negative.

$$cn(): -1\begin{bmatrix} \omega \\ \hat{\mathbf{c}} \end{bmatrix} = \begin{bmatrix} -\omega \\ -\hat{\mathbf{c}} \end{bmatrix}$$

Recap Lecture 9: Quaternion SLERP

The SLERP between two quaternions R_0 and R_1 can be calculated as

$$\frac{\sin[(1-\alpha)\Omega]}{\sin(\Omega)}\begin{bmatrix}\cos(\frac{\theta_0}{2})\\ \sin(\frac{\theta_0}{2})\hat{\mathbf{k}}_0\end{bmatrix}+\frac{\sin(\alpha\Omega)}{\sin(\Omega)}\begin{bmatrix}\cos(\frac{\theta_1}{2})\\ \sin(\frac{\theta_1}{2})\hat{\mathbf{k}}_1\end{bmatrix},$$

where Ω is the angle between the initial and final quaternions in R^n (i.e $R_0 \cdot R_1 = \cos(\Omega)$).

좀더 쉬운 버전?
오메가는 Quaternion angle

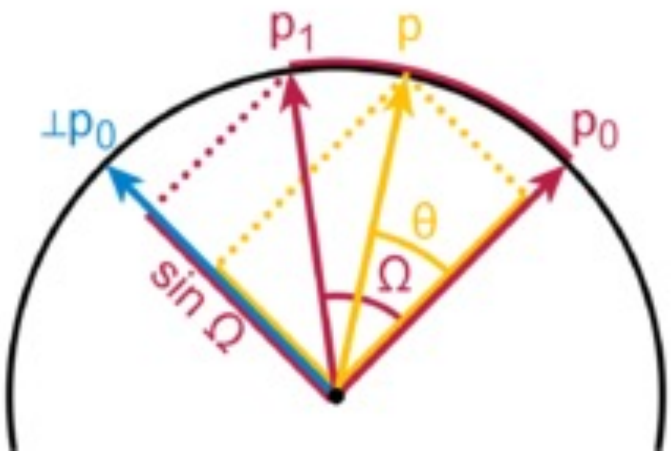


Image: https://en.wikipedia.org/wiki/Slerp

Rotation Interpolations

You can use either the power-based method or the quaternion SLERP to interpolate rotations.

- Power-based method:

$$R_\alpha := (R_1 R_0^{-1})^\alpha R_0$$

- Quaternion SLERP: 이거 쓰는거 추천?

$$\frac{\sin[(1-\alpha)\Omega]}{\sin(\Omega)}\begin{bmatrix}\cos(\frac{\theta_0}{2})\\ \sin(\frac{\theta_0}{2})\hat{\mathbf{k}}_0\end{bmatrix}+\frac{\sin(\alpha\Omega)}{\sin(\Omega)}\begin{bmatrix}\cos(\frac{\theta_1}{2})\\ \sin(\frac{\theta_1}{2})\hat{\mathbf{k}}_1\end{bmatrix}$$

Quaternion Angle

- When computing the quaternion angle, note that arccos and arcsin alone cannot determine the correct angle in the range of [0, 2**π**].
- For example, if we need to compute an angle ϕ given its cosine value p or sine value q, we can use the following formulas.
  - For the cosine value p, the arccos(p) results in $\pm\phi$.

$$\mathrm{p} = \cos\phi = \cos(-\phi) \qquad \arccos(p) = \phi \; or \; (-\phi)$$

  - For the sine value q, the arcsin(q) results in ϕ or (**π**- ϕ).

$$\mathrm{q} = \sin\phi = \sin(\pi - \phi) \qquad \arcsin(q) = \phi \; or \; (\pi - \phi)$$

Slide page credit: KAIST CS380 (Spring 2021) – Lab 6

Quaternion Angle

- To avoid the ambiguity, we recommend using the arctan function 이걸써라 instead of arccos or arcsin.
- The atan2 function returns the inverse tangent of a coordinate in radians. It is defined in the <cmath> header file.
- Mathematically, $atan2(y, x) = tan^{-1}(\frac{y}{x})$.

$$atan2\left(\sin(\phi), cos(\phi)\right) = \phi$$

Slide page credit: KAIST CS380 (Spring 2021) – Lab 6

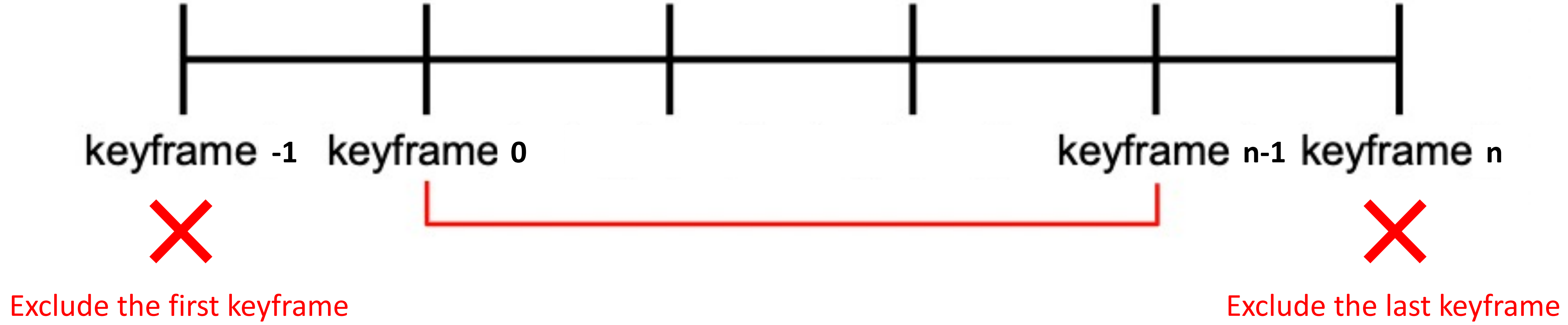
Task 3: Playing a Keyframe Animation

Task 3: Playing a Keyframe Animation

- Implement keyframe animation using the linear interpolation from Task 2.
- Hotkeys
  - ‘y’: Play/stop the animation.
  - ‘+’: Increase the animation speed.
  - ‘-’: Decrease the animation speed.

Animation by Interpolation

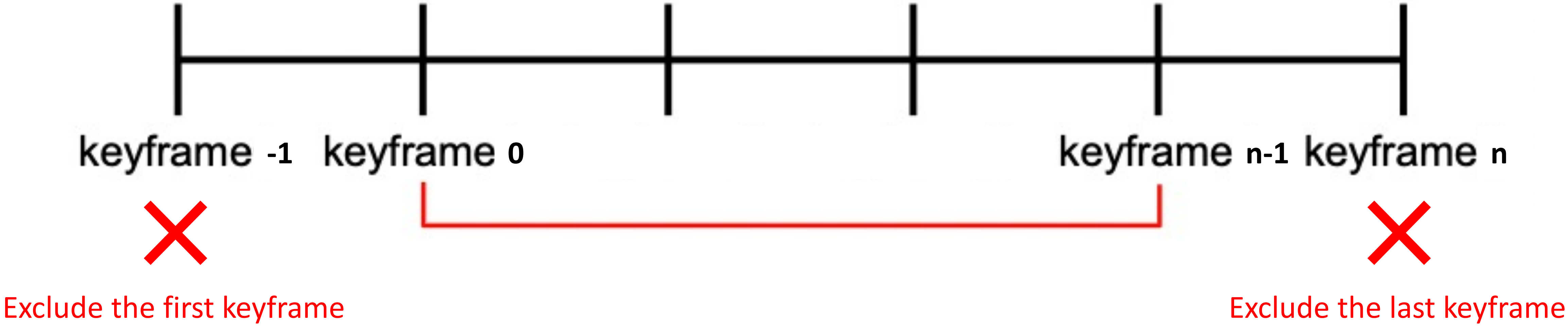
For the requirements of the next assignment (Assignment 6), interpolate the keyframes in the keyframe list, excluding the first and last keyframe.



Slide page credit: KAIST CS380 (Spring 2021) – Lab 6

Animation by Interpolation

When you have $(n+2)$ keyframes in the keyframe list $\{\text{keyframe}_i\}$ $(i \in [-1, n])$, only the middle n keyframes in $[0, n-1]$ will be used for the animation.

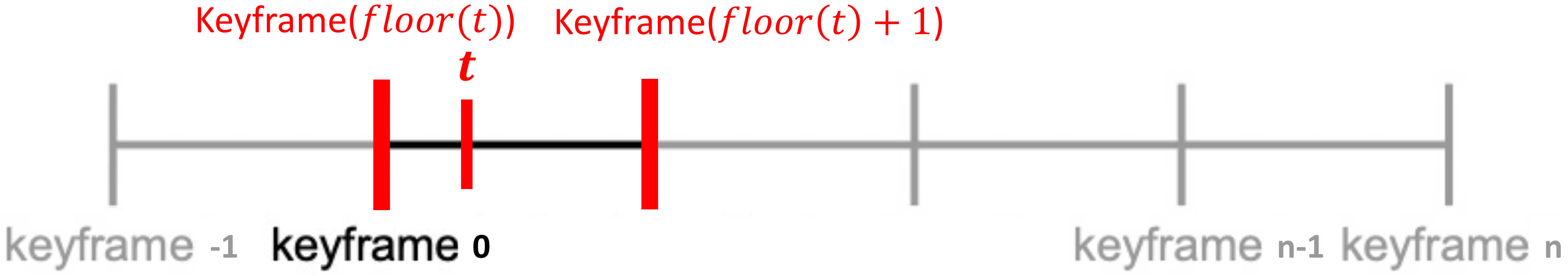


Interpolation of Two Frames

If we think of a real value t over the range $[0, n-1]$ it looks like below.

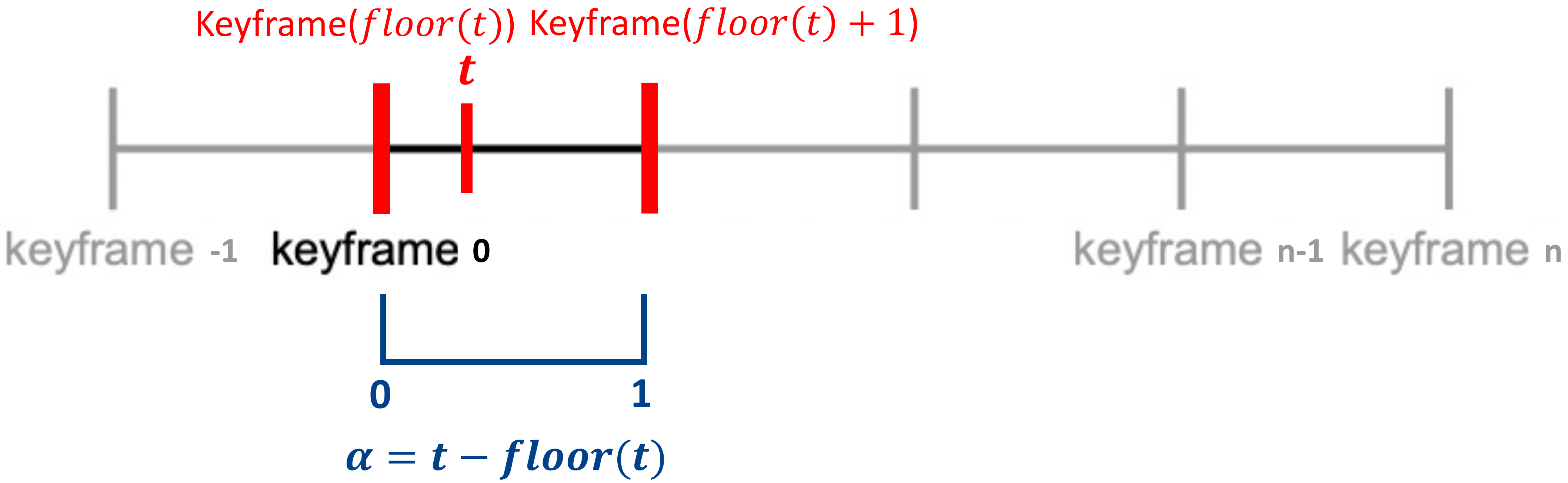
Interpolation of Two Frames

The closest left keyframe is the keyframe at index $floor(t)$, and the closest right keyframe is the keyframe at index $floor(t) + 1$.



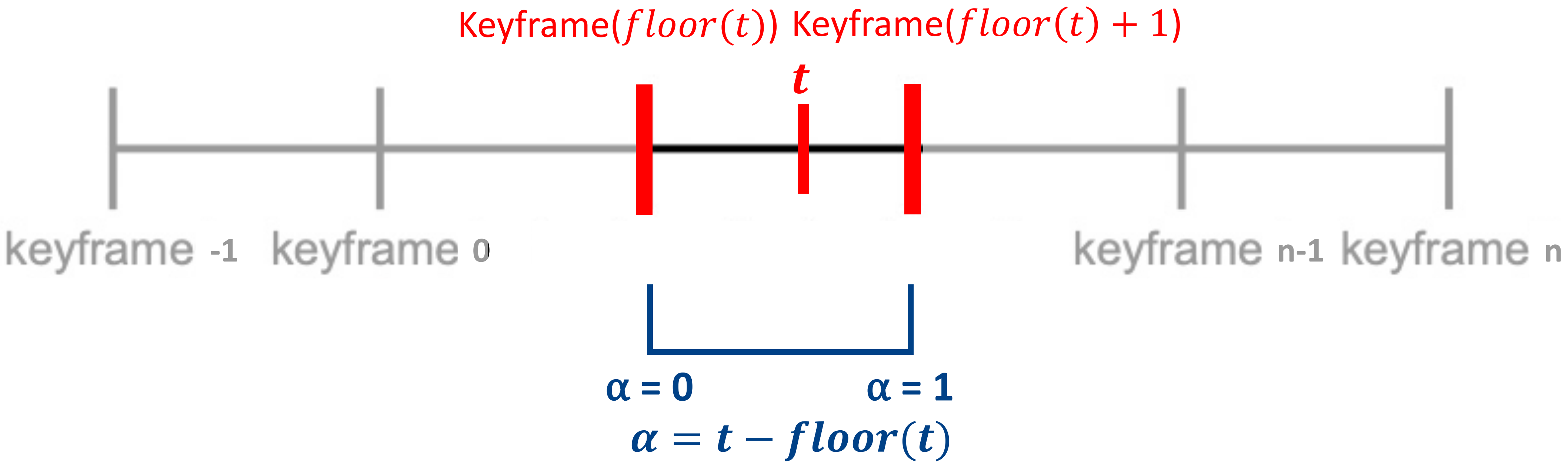
Interpolation of Two Frames

The interpolation factor α, which lies in the range $[0, 1]$, can be computed as t $- floor(t)$.



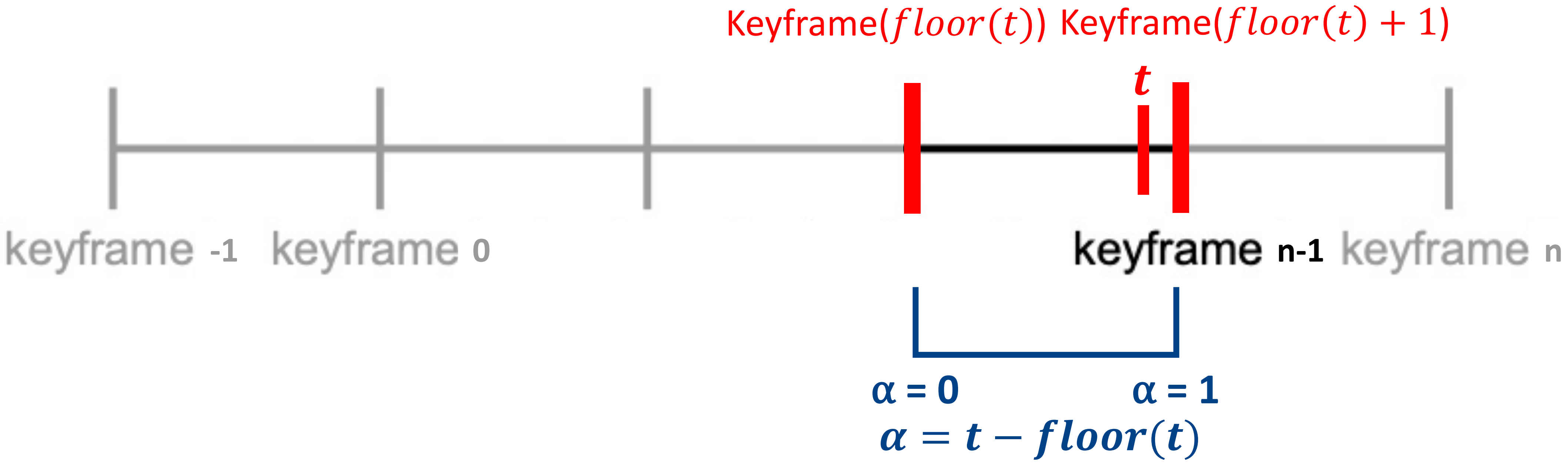
Interpolation of Two Frames

When frame $(i, i+1)$ interpolation is done, move to the next pair frame $(i+1, i+2)$ and interpolate with new α in $[0, 1]$.



Interpolation of Two Frames

Repeat until the end of the animation.



Playing Animation

void glutTimerFunc(time_ms, timerCallback, value)

- time_ms: time to call function (2^{nd} parameter)
- timerCallback: function to call
- value: argument for the callback function (2^{nd} parameter)

If you defined a frame interpolation function for animation, you can regularly call it at specific time intervals using glutTimerFunc.

Playing Animation

Example

```
void timerCallback(int next_ms) { … }
…
glutTimerFunc(time_ms, timerCallback, next_ms);
```

The timerCallback function (2nd parameter) will be executed in time_ms and the parameter next_ms will be passed to the callback function when it will be executed.

Playing Animation

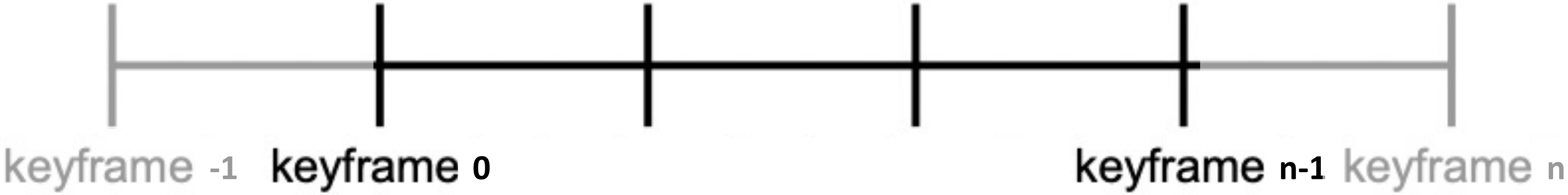
Let's implement the time callback function animateTimerCallback. Here is the snippet.

```
// Interpret "ms" as milliseconds into the animation
static void animateTimerCallback(int ms) {
    float t = (float)ms/(float)g_msBetweenKeyFrames;

    bool endReached = interpolateAndDisplay(t);
    if (!endReached)
        glutTimerFunc(1000/g_animateFramesPerSecond,
                      animateTimerCallback,
                      ms + 1000/g_animateFramesPerSecon);
    else { ... }
}
```

Playing Animation

Two global variables are required to control the speed of the animation play.

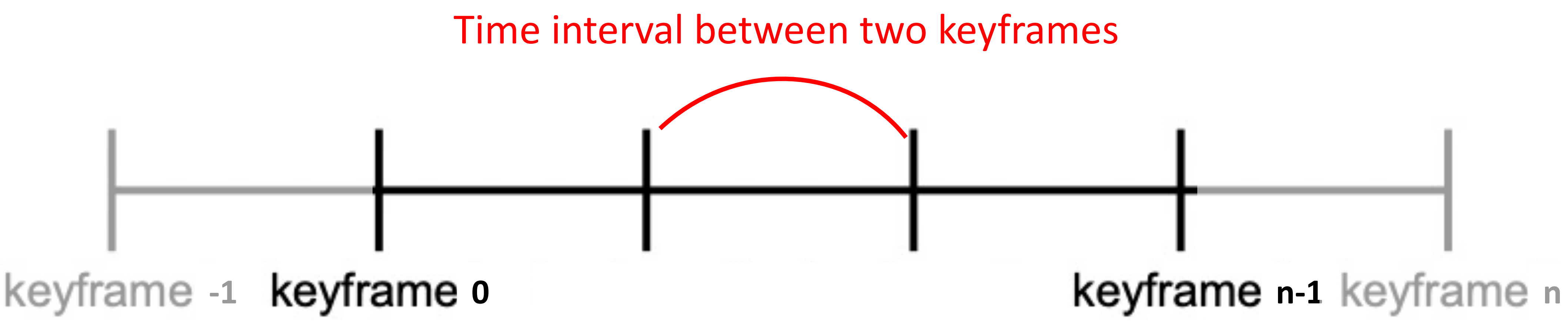
```
static int g_msBetweenKeyFrames = 2000;
static int g_animateFramesPerSecond = 60;
```



Playing Animation

Two global variables are required to control the speed of the animation play.

```
static int g_msBetweenKeyFrames = 2000;
```

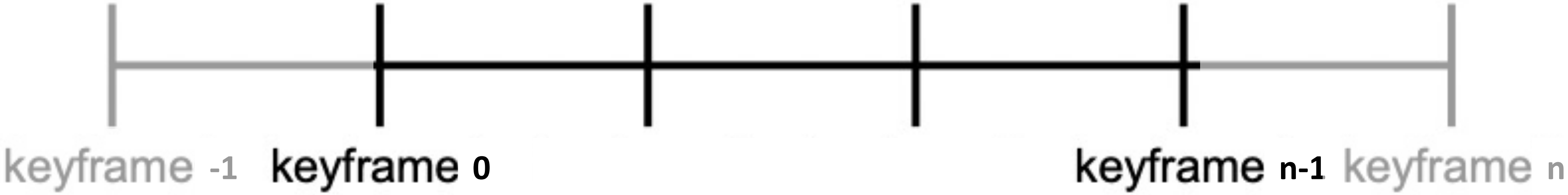


Playing Animation

Two global variables are required to control the speed of the animation play.

```
static int g_animateFramesPerSecond = 60;
```

The number of frames rendered in 1 second



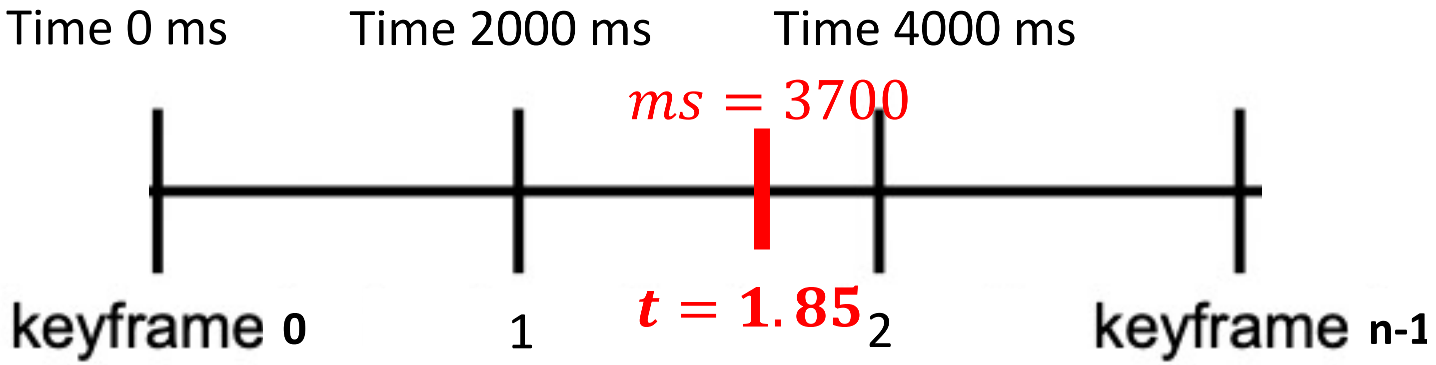
Playing Animation

- Here, the argument ms represents the time interval in milliseconds since the last call to the timer callback function.
- We can compute t in $[0, n-1]$ using the argument ms like below.

```
// Interpret "ms" as milliseconds into the animation
static void animateTimerCallback(int ms) {
    float t = (float)ms/(float)g_msBetweenKeyFrames;

    bool endReached = interpolateAndDisplay(t);
    if (!endReached)
        glutTimerFunc(1000/g_animateFramesPerSecond,
                      animateTimerCallback,
                      ms + 1000/g_animateFramesPerSecon);
    else { ... }
}
```

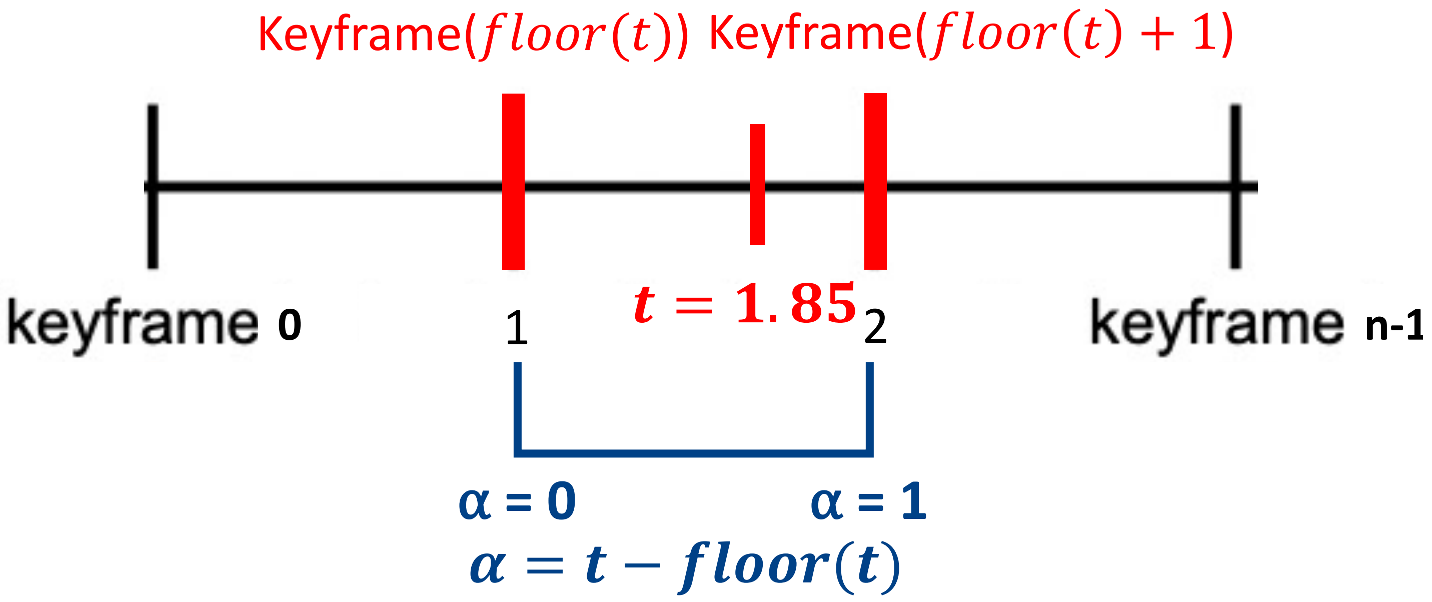
Playing Animation

- For example, ms is $3700ms$ and g_msBetweenKeyFrames is 2000.
- Then $t = \frac{ms}{g_{msBetweenKeyFrames}} = \frac{3700}{2000} = 1.85$.



Playing Animation

Using this t, we can compute the alpha and interpolate the surrounding frames; keyframe($floor(t)$) and keyframe($floor(t) + 1$).



Playing Animation

Each time you interpolate the frame, you need to return the Boolean value that tells whether the animation has reached the end or not.

```
// Interpret "ms" as milliseconds into the animation
static void animateTimerCallback(int ms) {
    float t = (float)ms/(float)g_msBetweenKeyFrames;

    bool endReached = interpolateAndDisplay(t);
    if (!endReached)
        glutTimerFunc(1000/g_animateFramesPerSecond,
                      animateTimerCallback,
                      ms + 1000/g_animateFramesPerSecon);
    else { ... }
}
```

Playing Animation

If it is false, you need to call the callback function again (recursively) with the new ms value. The next interpolation should happen in $\left(\frac{1000}{\text{g\_animateFramesPerSecon}}\right)$.

```
// Interpret "ms" as milliseconds into the animation
static void animateTimerCallback(int ms) {
    float t = (float)ms/(float)g_msBetweenKeyFrames;

    bool endReached = interpolateAndDisplay(t);
    if (!endReached)
        glutTimerFunc(1000/g_animateFramesPerSecond,
                      animateTimerCallback,
                      ms + 1000/g_animateFramesPerSecon);
    else { ... }
}
```

Playing Animation

- If it is true, stop the animation.
- After the animation is finished, set the current keyframe as $(n-1)th$ keyframe among $[-1, n]$ keyframes.

```
// Interpret "ms" as milliseconds into the animation
static void animateTimerCallback(int ms) {
    float t = (float)ms/(float)g_msBetweenKeyFrames;

    bool endReached = interpolateAndDisplay(t);
    if (!endReached)
        glutTimerFunc(1000/g_animateFramesPerSecond,
                      animateTimerCallback,
                      ms + 1000/g_animateFramesPerSecon);
    else { ... }
}
```

Animation Speed Control

- You can adjust the speed of the animation by changing the value of g_msBetweenKeyFrames.
- Change the value of g_msBetweenKeyFrames by incrementing or decrementing 100ms, respectively, for each press of the ‘+’ or ‘-’ key.
- Note that g_msBetweenKeyFrames cannot have a negative value.

Animation on Display

Each time you obtain the intermediate frames by interpolation.

- Don’t forget to set the interpolated frame on your scene graph’s nodes.
- Don’t forget to call glutPostRedisplay() after setting the frames to trigger a redraw of the display window and show the updated animation to the user.

Evaluation

- Are the keyframe list’s hotkeys working well?
- Is implementation in interpolation.h TODOs correct?
- Are the animation action’s hotkeys working well?

Submission

- Due: Sun, May 14 23:59 KST.
- Late submission: Up to two days (~Tues, May 16 23:59 KST) with a penalty of 20% of the score.
- Submit on the GradeScope in a zip file.
- Do not need to include the tutorial files in your submission (list_tutorial.cpp).